C++ DESIGN PATTERNS AND DERIVATIVES PRICING

Mathematics, Finance and Risk

Editorial Board

C++ DESIGN PATTERNS AND DERIVATIVES PRICING

MARK S. JOSHI
Royal Bank of Scotland Group

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011–4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published 2004

Printed in the United Kingdom at the University Press, Cambridge

Typeface Times 11/14 pt. *System* LaTeX 2$_\varepsilon$ [TB]

A catalogue record for this book is available from the British Library

Library of Congress Cataloguing in Publication data available

Joshi, M.S. (Mark Suresh), 1969–
C++ design patterns and derivatives pricing / M.S. Joshi.
p. cm. – (Mathematics, finance, and risk)
Includes bibliographical references and index.
ISBN 0 521 83235 7
1. Derivative securities–Prices–Mathematical models. 2. C++ (Computer program language)
3. Business mathematics. I. Title. II. Series.
HG6024.A3J665 2004
332.64′57′02855133 – dc22 2003065293

ISBN 0 521 83235 7 hardback

To Jane

Contents

Preface *page* xi
1 A simple Monte Carlo model 1
1.1 Introduction 1
1.2 The theory 1
1.3 A simple implementation of a Monte Carlo call option pricer 2
1.4 Critiquing the simple Monte Carlo routine 7
1.5 Identifying the classes 9
1.6 What will the classes buy us? 10
1.7 Why object-oriented programming? 11
1.8 Key points 11
1.9 Exercises 12
2 Encapsulation 13
2.1 Implementing the pay-off class 13
2.2 Privacy 15
2.3 Using the pay-off class 16
2.4 Further extensibility defects 19
2.5 The open-closed principle 20
2.6 Key points 21
2.7 Exercises 22
3 Inheritance and virtual functions 23
3.1 'is a' 23
3.2 Coding inheritance 24
3.3 Virtual functions 24
3.4 Why we must pass the inherited object by reference 29
3.5 Not knowing the type and virtual destruction 30
3.6 Adding extra pay-offs without changing files 34
3.7 Key points 37
3.8 Exercises 37

4 Bridging with a virtual constructor 38
4.1 The problem 38
4.2 A first solution 39
4.3 Virtual construction 43
4.4 The rule of three 51
4.5 The bridge 53
4.6 Beware of new 57
4.7 A parameters class 58
4.8 Key points 64
4.9 Exercises 64
5 Strategies, decoration and statistics 65
5.1 Differing outputs 65
5.2 Designing a statistics gatherer 65
5.3 Using the statistics gatherer 68
5.4 Templates and wrappers 72
5.5 A convergence table 76
5.6 Decoration 79
5.7 Key points 80
5.8 Exercises 80
6 A random numbers class 82
6.1 Why? 82
6.2 Design considerations 83
6.3 The base class 85
6.4 A linear congruential generator and the adapter pattern 87
6.5 Anti-thetic sampling via decoration 92
6.6 Using the random number generator class 96
6.7 Key points 101
6.8 Exercises 101
7 An exotics engine and the template pattern 102
7.1 Introduction 102
7.2 Identifying components 103
7.3 Communication between the components 104
7.4 The base classes 106
7.5 A Black–Scholes path generation engine 110
7.6 An arithmetic Asian option 114
7.7 Putting it all together 116
7.8 Key points 119
7.9 Exercises 119

8 Trees 121
8.1 Introduction 121
8.2 The design 123
8.3 The `TreeProduct` class 125
8.4 A tree class 129
8.5 Pricing on the tree 135
8.6 Key points 139
8.7 Exercises 139
9 Solvers, templates and implied volatilities 141
9.1 The problem 141
9.2 Function objects 142
9.3 Bisecting with a template 145
9.4 Newton–Raphson and function template arguments 149
9.5 Using Newton–Raphson to do implied volatilities 151
9.6 The pros and cons of templatization 154
9.7 Key points 156
9.8 Exercises 156
10 The factory 157
10.1 The problem 157
10.2 The basic idea 157
10.3 The singleton pattern 158
10.4 Coding the factory 159
10.5 Automatic registration 162
10.6 Using the factory 165
10.7 Key points 166
10.8 Exercises 167
11 Design patterns revisited 168
11.1 Introduction 168
11.2 Creational patterns 168
11.3 Structural patterns 169
11.4 Behavioural patterns 170
11.5 Why design patterns? 171
11.6 Key points 172
11.7 Further reading 172
11.8 Exercises 173
Appendix A Black–Scholes formulas 174
Appendix B Distribution functions 178
Appendix C A simple array class 182

C.1 Choosing an array class 182
C.2 The header file 183
C.3 The source code 186
Appendix D The code 193
D.1 Using the code 193
D.2 Compilers 193
D.3 License 193
Appendix E Glossary 194
Bibliography 195
Index 197

Preface

This book is aimed at a reader who has studied an introductory book on mathematical finance and an introductory book on C++ but does not know how to put the two together. My objective is to teach the reader not just how to implement models in C++ but more importantly how to think in an object oriented way. There are already many books on object-oriented programming, however, the examples tend not to feel real to the financial mathematician so in this book we work exclusively with examples from derivatives pricing.

We do not attempt to cover all sorts of financial models but instead examine a few in depth with the objective at all times of using them to illustrate certain OO ideas. We proceed largely by example and our design process is not one to be recommended because we rewrite our designs as new concepts are introduced, instead of working out a great design at the start. Whilst this approach is not optimal from a design standpoint, it is more pedagogically accessible. An aspect of this is that our examples are designed to emphasize design principles rather than to illustrate other features of coding such as numerical efficiency or exception safety.

We commence by introducing a simple Monte Carlo model which does not use OO techniques but rather is the simplest procedural model for a pricing a call option one could write. We examine its shortcomings and discuss how classes naturally arise from the concepts involved in its construction.

In Chapter 2, we move on to the concept of encapsulation – the idea that a class allows to express a real-world analogue and its behaviours precisely. In order to illustrate encapsulation, we look at how a class to define the pay-off of a vanilla option could be defined. We also see that the class we have defined has certain defects, and this naturally leads on to the open-closed principle.

In Chapter 3, we see how a better pay-off class can be defined by using inheritance and virtual functions. This raises technical issues involving destruction and passing arguments which we address. We also see how this approach is compatible with the open–closed principle.

Using virtual functions causes problems regarding the copying of objects of unknown type, and in Chapter 4 we address these problems. We do so by introducing virtual constructors and the bridge pattern. We digress to discuss the 'rule of three' and the slowness of `new`. The ideas are illustrated via a vanilla options class and a parameters class.

With these new techniques at our disposal, we move on to looking at more complicated design patterns in Chapter 5. We first introduce the strategy pattern that express the idea that decisions on part of an algorithm can be deferred by delegating responsibilities to an auxiliary class. We then look at how templates can be used to write a wrapper class that removes a lot of our difficulties with memory handling. As an application of these techniques, we develop a convergence table using the decorator pattern.

In Chapter 6, we look at how to develop a random numbers class. We first examine why we need a class and then develop a simple implementation which provides a reusable interface and an adequate random number generator. We use the implementation to introduce and illustrate the adapter pattern, and to examine further the decorator pattern.

We move on to our first non-trivial application in Chapter 7, where we use the classes developed so far in the implementation of a Monte Carlo pricer for path-dependent exotic derivatives. As part of this design, we introduce and use the template pattern. We finish with the pricing of Asian options.

We shift from Monte Carlo to trees in Chapter 8. We see the similarities and differences between the two techniques, and implement a reusable design. As part of the design, we reuse some of the classes developed earlier for Monte Carlo.

We return to the topic of templates in Chapter 9. We illustrate their use by designing reusable solver classes. These classes are then used to define implied volatility functions. En route, we look at function objects and pointers to member functions. We finish with a discussion of the pros and cons of templatization.

In Chapter 10, we look at our most advanced topic: the factory pattern. This patterns allows the addition of new functionality to a program without changing any existing files. As part of the design, we introduce the singleton pattern.

Our final chapter reviews, assesses and organizes the patterns we have covered. In particular, we classify them as creational, structural and behavioural patterns. We finish with a list of further books to read.

All the code in this book is included on the accompanying CD. The book's website is

`www.markjoshi.com/design`

and any bugfixes will be posted there.

Acknowledgements

I am grateful to the Royal Bank of Scotland for providing a stimulating environment in which to learn, study and do mathematical finance. Most of my views on coding C++ and financial modelling have been developed during my time working there. My understanding of the topic has been formed through daily discussions with current and former colleagues including Chris Hunter, Peter Jäckel, Dherminder Kainth, Sukhdeep Mahal, Robin Nicholson and Jochen Theis. I am also grateful to a host of people for their many comments on the manuscript, including Alex Barnard, Dherminder Kainth, Rob Kitching, Sukhdeep Mahal, Nadim Mahassen, Hugh McBride, Alan Stacey and Patrik Sundberg. I would also like to thank David Tranah and the rest of the team at Cambridge University Press for their careful work and attention to detail. Finally my wide has been very supportive.